IF: a Tool-set for validation of distributed real-time systems

Susanne Graf

Marius Bozga, Laurent Mounier,
Yassine Lakhnech, Joseph Sifakis

VERIMAG
Grenoble

Theory, methods and tools for design and validation of distributed and safety critical systems

- Synchronous languages, development of embedded systems
  - **Lustre** language: compilation, verification and test
    → Telelogic **SCADE**
- Tools and methods based on timed and hybrid automata
  - synthesis and validation of schedulers and controllers
  - **Kronos** tool for the verification of timed systems
- Tools and methods for communication systems
  - Semantics and real-time extensions of design languages
  - Verification of security protocols
  - Validation tools: **Xesar**, **CADP**, **TGV**, **Invest**, **IF**

Motivation

Goal

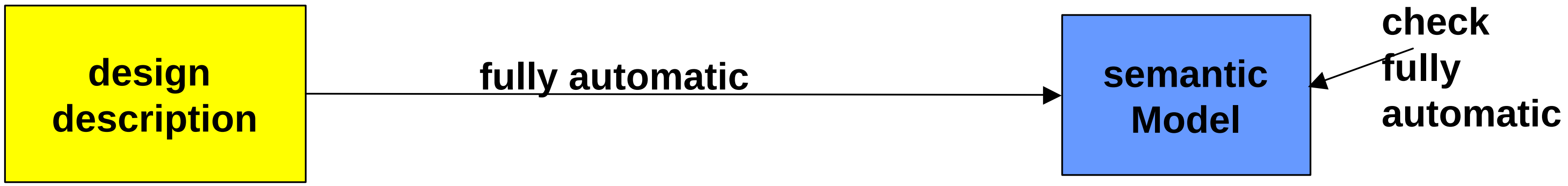
Combine state-of-the-art validation with commercial development tools

Context

Telecommunication systems,
Real-time embedded systems

Model-checking: its problems

The idea: why MC is attractive



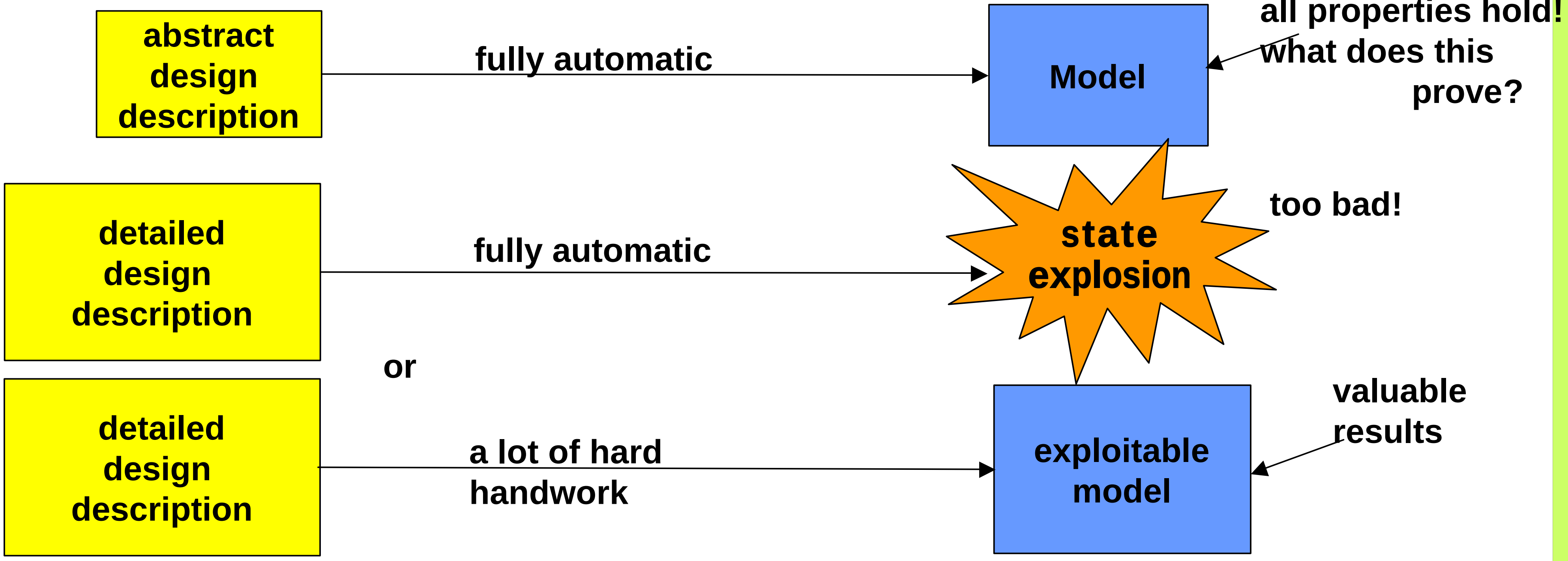
design description

fully automatic

semantic Model

check fully automatic

The reality: why has MC a bad reputation



abstract design description

fully automatic

Model

all properties hold! what does this prove?

detailed design description

fully automatic

state explosion

too bad!

or

detailed design description

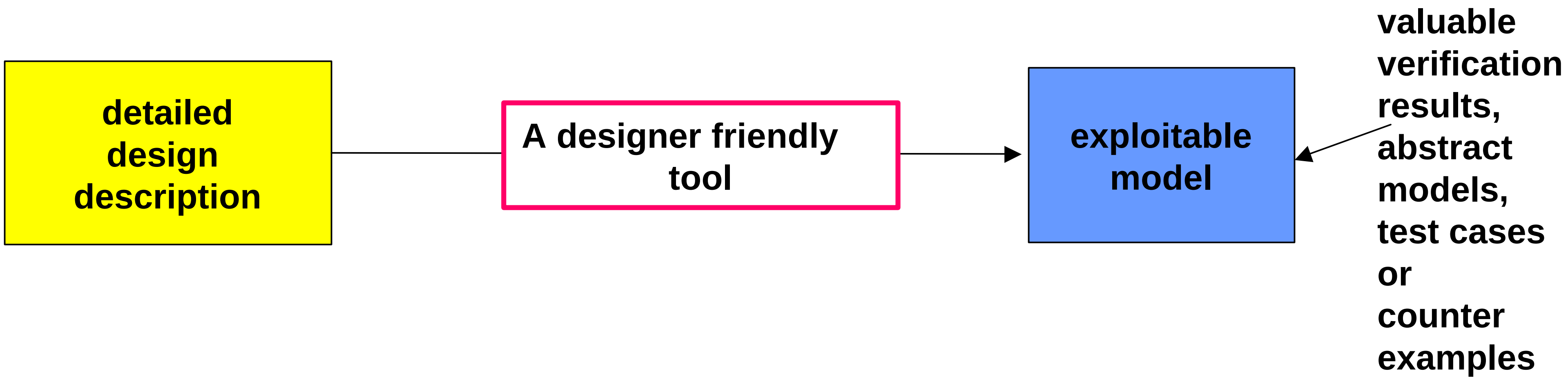
a lot of hard handwork

exploitable model

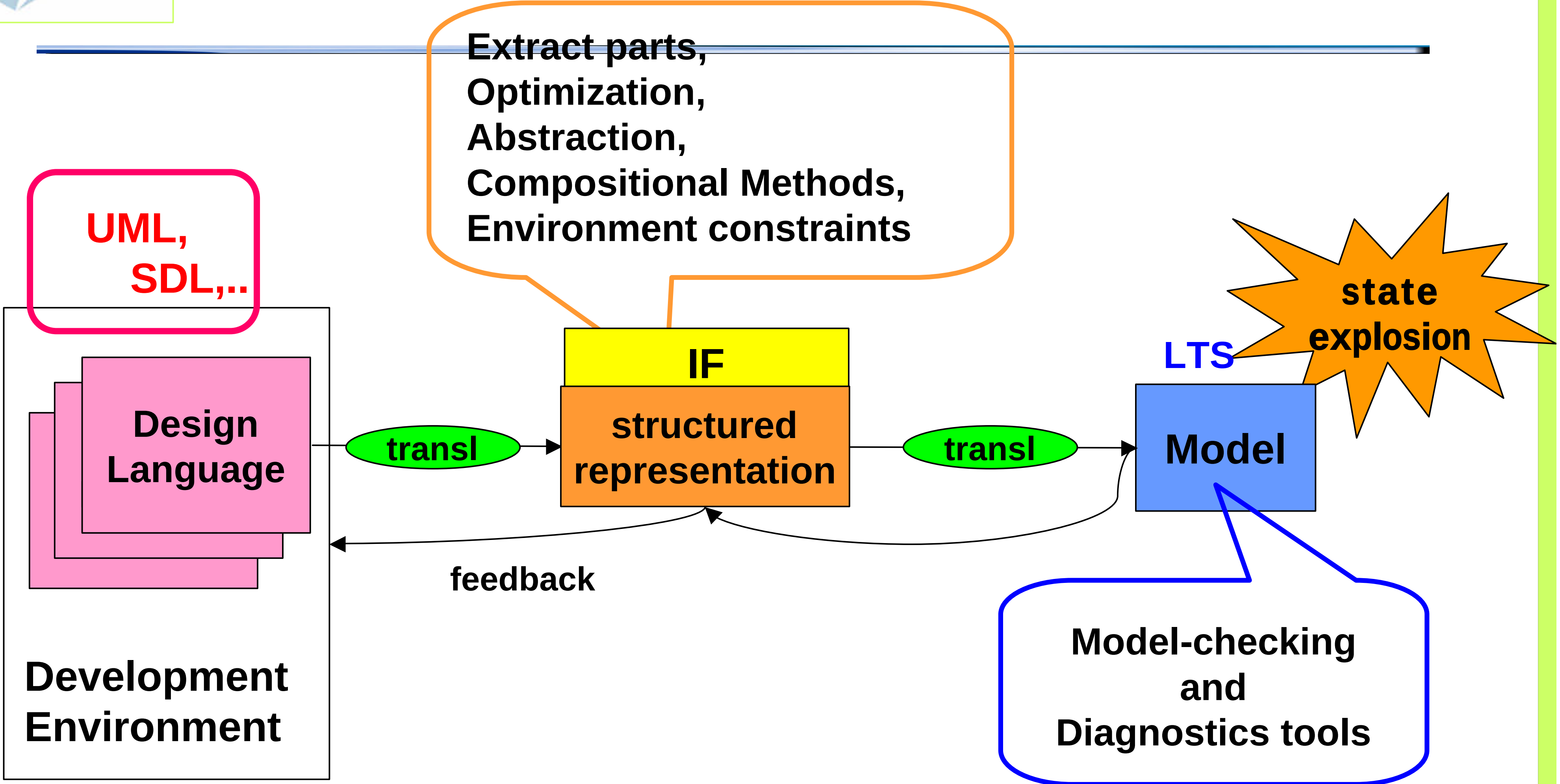
valuable results

Model-checking: how it should be

This bad reputation must not be justified


detailed design description
A designer friendly tool
exploitable model
valuable verification results, abstract models, test cases or counter examples

Principle of a validation environment



Need for a structured system representation

1. Intermediate and tool exchange format
   - Basis for static analysis, abstraction and compositional methods
   - Connection of a large range of high level design languages of with analysis tools (model-checking, performence,...)
   - Exchange of structured system descriptions between analysis tools
2. Study of time models
   - Need for a appropriate time extensions of languages for communicating and distributed systems (SDL, UML)
   - Appropriate for design and verification of real-time systems

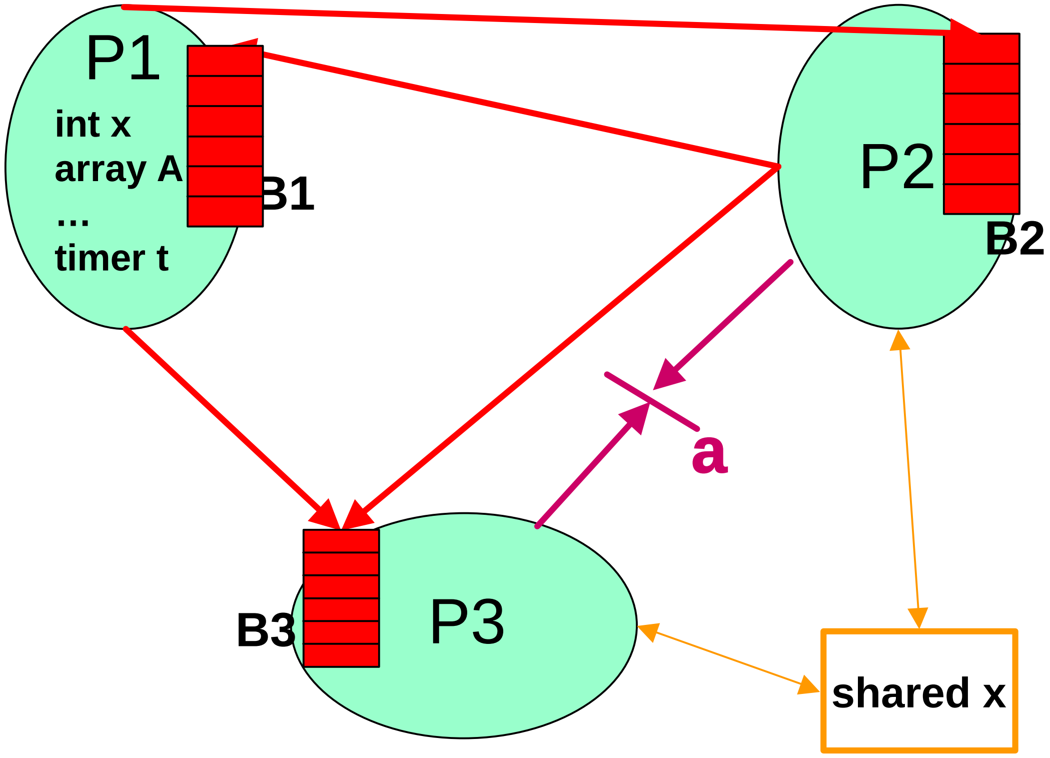
Outline

1. Motivations
2. IF intermediate representation
3. IF validation tool-set
4. Case studies
5. Conclusions

2. IF intermediate representation

System structure at instant t

Communicating extended timed automata (with urgency)



Communication/ Interaction

- **asynchronous** channels
  (reliable?, bounded?, delay?)
- **synchronous** rendez-vous
- **shared** variables

Time representation

Timed automata with

Urgency of transitions

(eager,lazy, delayable)

IF: Processes

- A set of local variables
  - elementary: bool, int, ... timers and clocks, ...
  - structured: array, record
  - abstract
- A set of control states with attributes:
  - stable/nostable (control observable states)
  - save and discard sets (reordering of input message buffer)
- A set of control transitions: $\mathbf{s} \xrightarrow[\textbf{urgency ; priority}]{\textbf{guard} \rightarrow \textbf{input ; body}} \mathbf{s'}$

IF: Transitions (abstract syntax)

$$s \xrightarrow[\text{urgency ; priority}]{\text{guard} \rightarrow [\text{ input }] ; [\text{ body }]} s'$$

$$s \xrightarrow[\text{urgency ; priority}]{\text{guard} \rightarrow \text{sync}} s'$$

- guard: boolean expression on data, timers, clocks
- input: asynchronous message inputs from buffers
- sync: gate synchronization

E

- body: action*
  - asynchronous message outputs to buffers
  - re/setting of timers/clocks
  - assignments
  - complex instructions

A

- urgency attribute: **eager, lazy, delayable**
- priority

Pr

Timed automata with urgency [BornotSifakis97]

- System transitions take 0 time (assimilated with an *event* "transition started", "transition terminated", ...) & time progresses in states, measured by clocks and timers

- **Urgency** defines when **enabled** system transitions are taken
  - **enabled** eager transitions are urgent, that is terminated « now » (or disabled by other system transitions)
  - **enabled** lazy transitions are never urgent, that means they can be disabled by time-progress
  - **enabled** delayable transitions are not disabled by time-progress, but it is taken for granted that they will be taken (except if disabled by other system transitions)

Timed automata with urgency

Allow to express a rich spectrum of time paradigms

1. Totally asynchronous view (no assumption on time progress): all transitions are lazy

   Ensure safe behaviour despite violation of deadlines

2. Synchronous view (next tick/input when system has finished): all transitions are eager

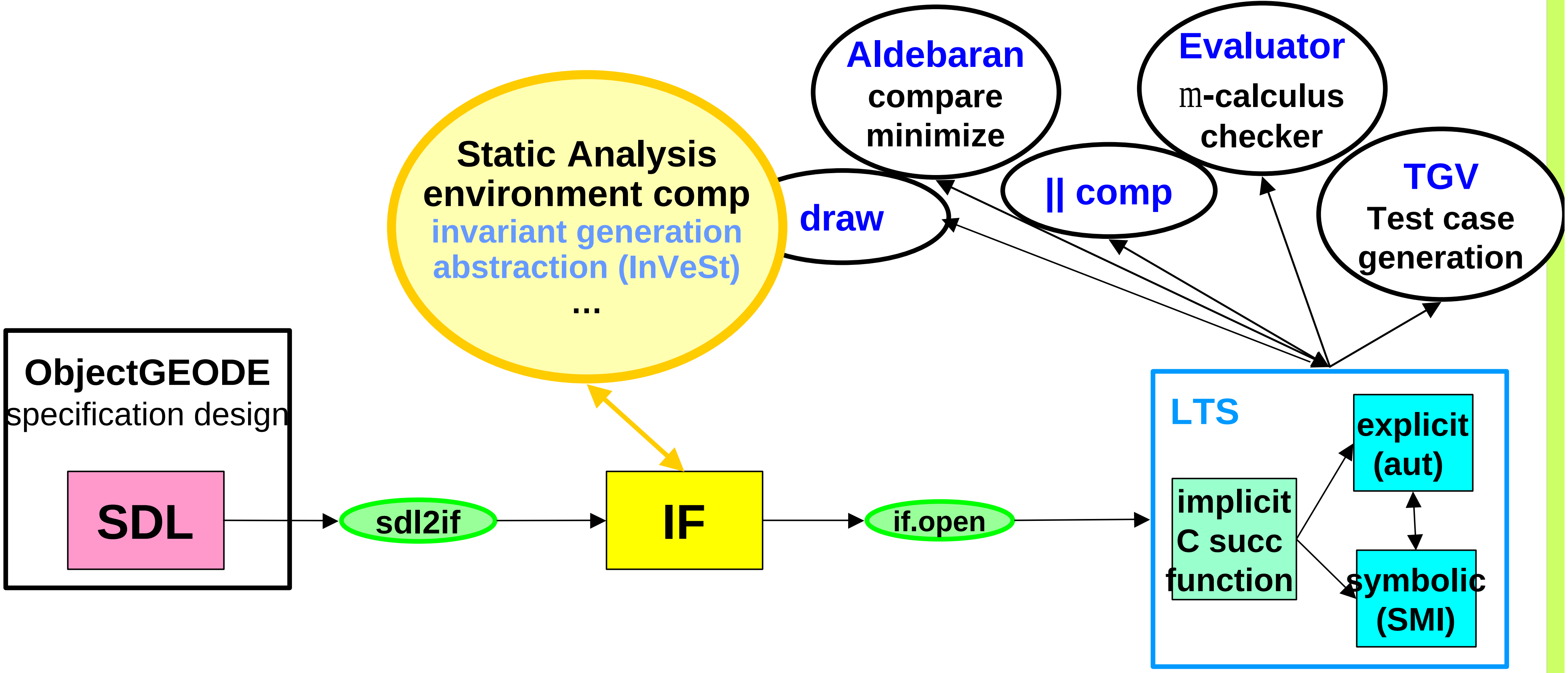
   Ensure safe behaviour under strong assumptions (risk of time-lock)

3. Real-time views: different urgency types and time guards:

Outline

1. Motivations
2. IF intermediate representation
3. IF validation tool-set
4. Applications
5. Conclusion and perspectives

3. Architecture of the IF tool-set



Translation from SDL to IF: sdl2if

- Based on an ObjectGeode API ⇨

we follow standard evolution of SDL

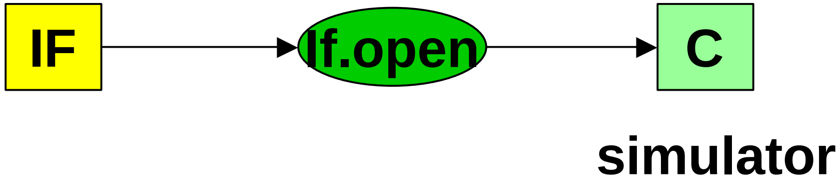
- Supports almost all of SDL'96:
  - timeouts are translated by time-guards
  - elimination of block hierarchy ("flat" architecture)
  - destination of outputs is statically determined if possible (only delaying channels represented explicitly)

  Only for more efficient verification

  - procedures are inlined (no recursion allowed)

Translation IF to LTS

Simulator construction: if.open



- implements:
  - discrete/dense time
  - partial order reduction
  - compositional generation
- supports:
  - random/guided simulation
  - on-the-fly verification
  - explicit LTS construction

LTS level validation components

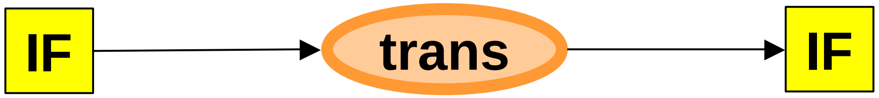
- Basic Functionalities
  - switch representations
  - parallel composition
  - draw graphical representations (valid property)
  - generate MSCs from (diagnostic) sequence (invalid property)

- Model-checking:
  - temporal-logic properties (**Evaluator, Kronos**)
  - behavioral comparison and reduction (**Aldebaran**)

(both including diagnostic capabilities)

- Test case generation (**TGV**)

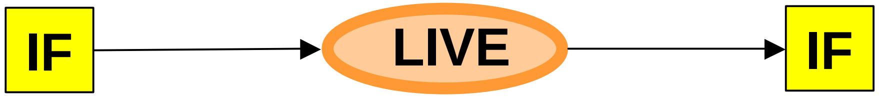
Static Analysis and Abstraction



PRINCIPLE

- Source code transformation in order to get
  - a smaller state representation
  - less states, which represent sets
- Preserve a set of (safety) properties (strongly or weakly)
- Combine several static analysis methods

Static Analysis and Abstraction (property independant)



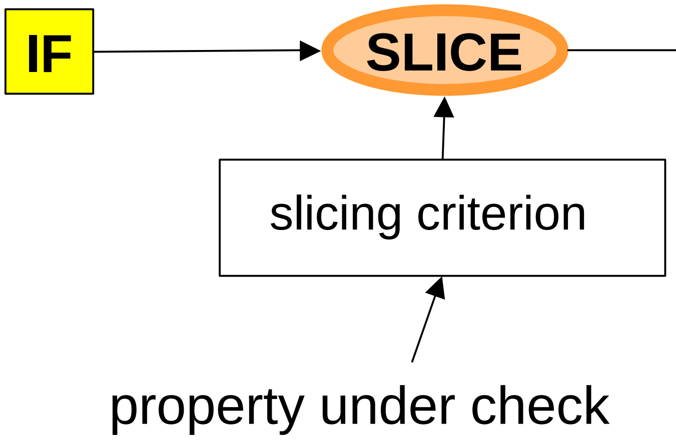
Live variable analysis

and

constant elimination

- Reset all live variables not live in some control point (its value is irrelevant in this state)
- Invalidate non-live clocks (clock reduction)
- Eliminate globally dead variables
- Replace constants by their value

Static Analysis and Abstraction (property dependant)



Slicing

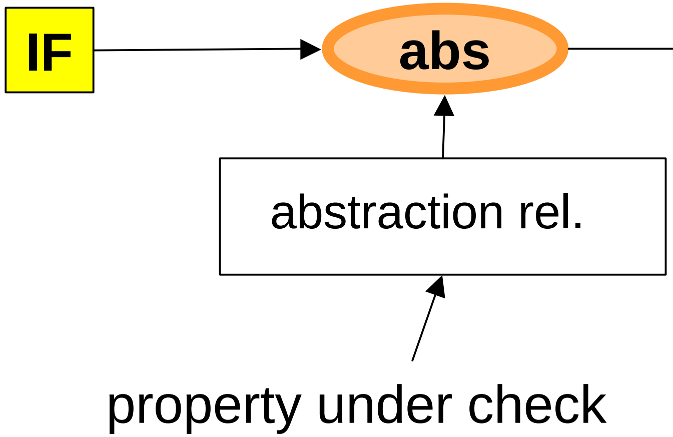
observables: messages, variables, …
(in particular control states)

test purpose, abstract behaviour,
temporal logic formula…

- Eliminate non relevant parts of the system with respect to a **slicing criterion**
  (variables, messages, transitions, processes)

example

Static Analysis and Abstraction (property dependant)



abstraction

(variable elimination, data abstraction, predicate abstraction (InVeST), …)

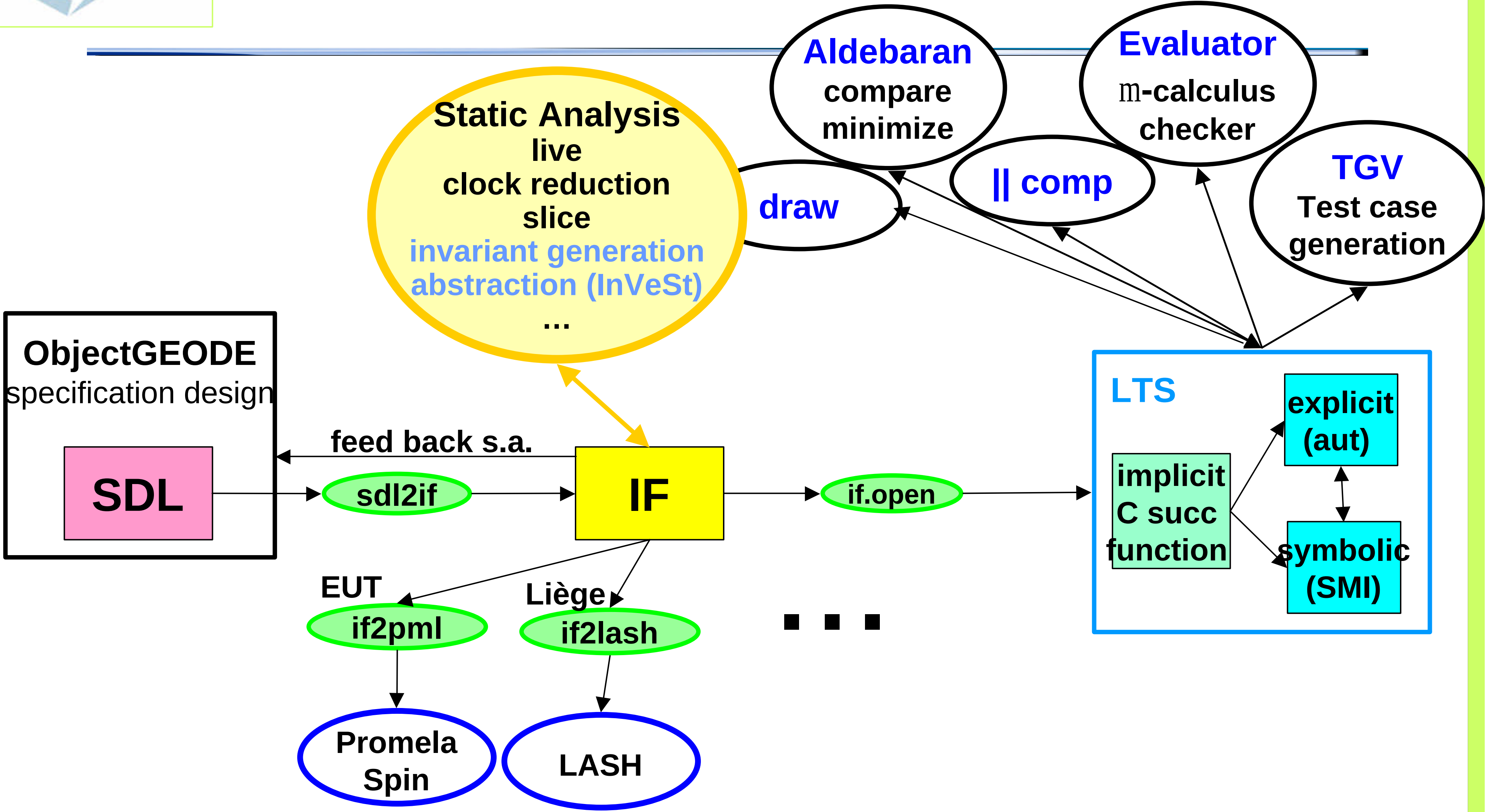
(test purpose, observer, TL property…)

Static Analysis and Abstraction

Summary

- In practice: drastic reductions of the state graph
- "abstract program" computed, can be directly used by other tools
- Notice:
  - static analyses and abstractions can be combined, preserving the intersection of the properties
  - abstraction means (in general) weak preservation of properties

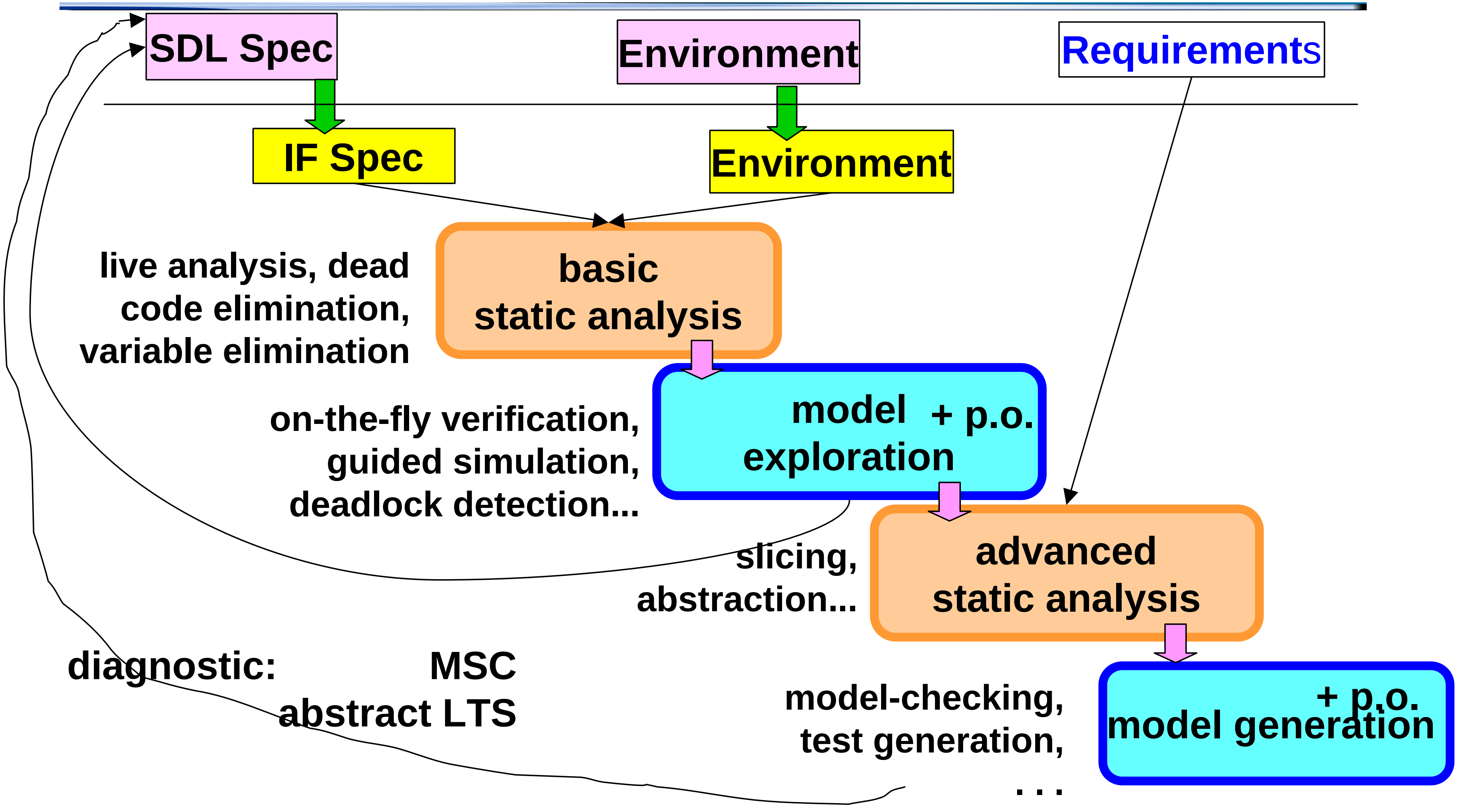
Architecture of the IF tool-set



Outline

1. Motivations
2. IF intermediate representation
3. IF validation and test generation environment
4. Applications
5. Conclusion and perspectives

Validation methodology


SDL Spec
Environment
Requirements
IF Spec
Environment
basic static analysis
live analysis, dead code elimination, variable elimination
model exploration + p.o.
on-the-fly verification, guided simulation, deadlock detection...
advanced static analysis
slicing, abstraction...
model generation + p.o.
model-checking, test generation, ...
diagnostic: MSC abstract LTS

Validation methodology: taking into account environment constraints

Open systems: environment constraints (EC) are essential for successful verification

verification results

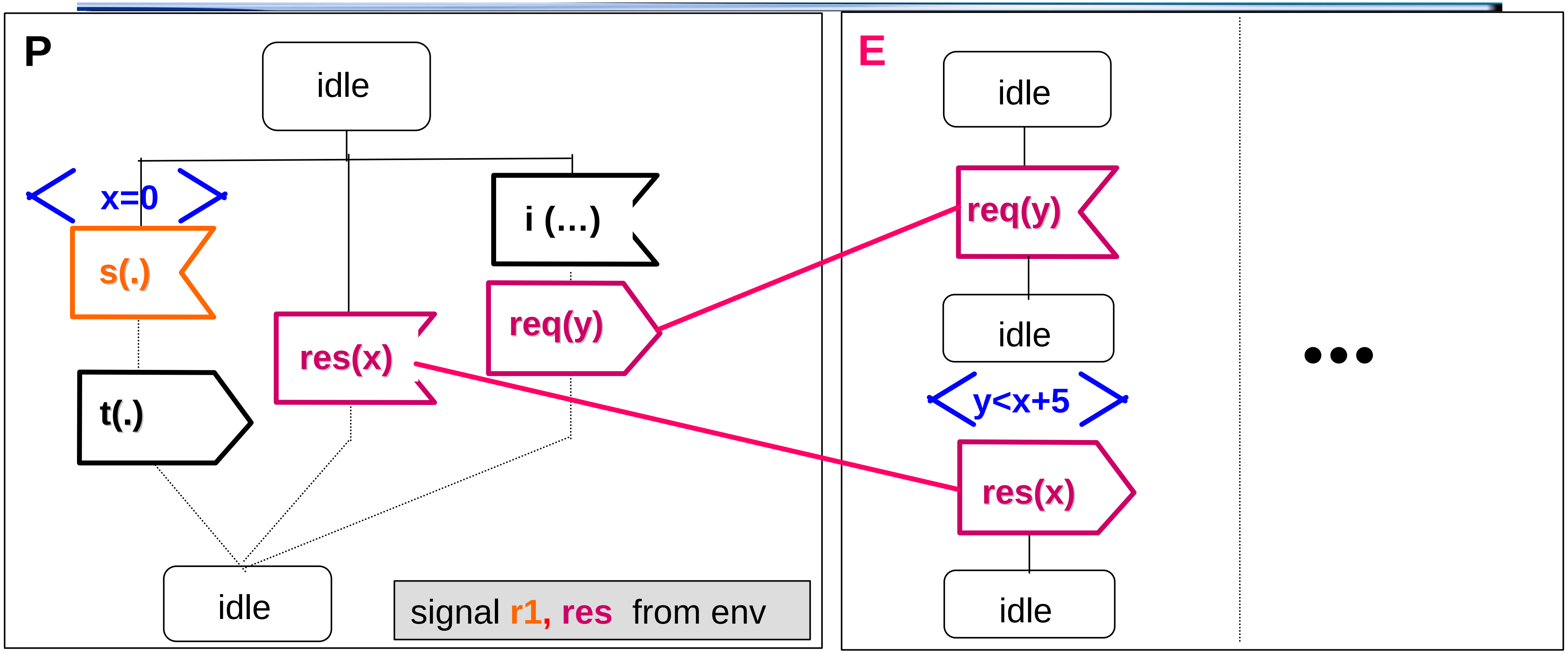
Sys |= EC => P

- Solution: describe EC by a (set of) processes **E**

  Verify the **Sys || E,**

  where **Sys** and **E** communicate by synchronous rendez-vous

Environment constraints (example)



Environment constraints:

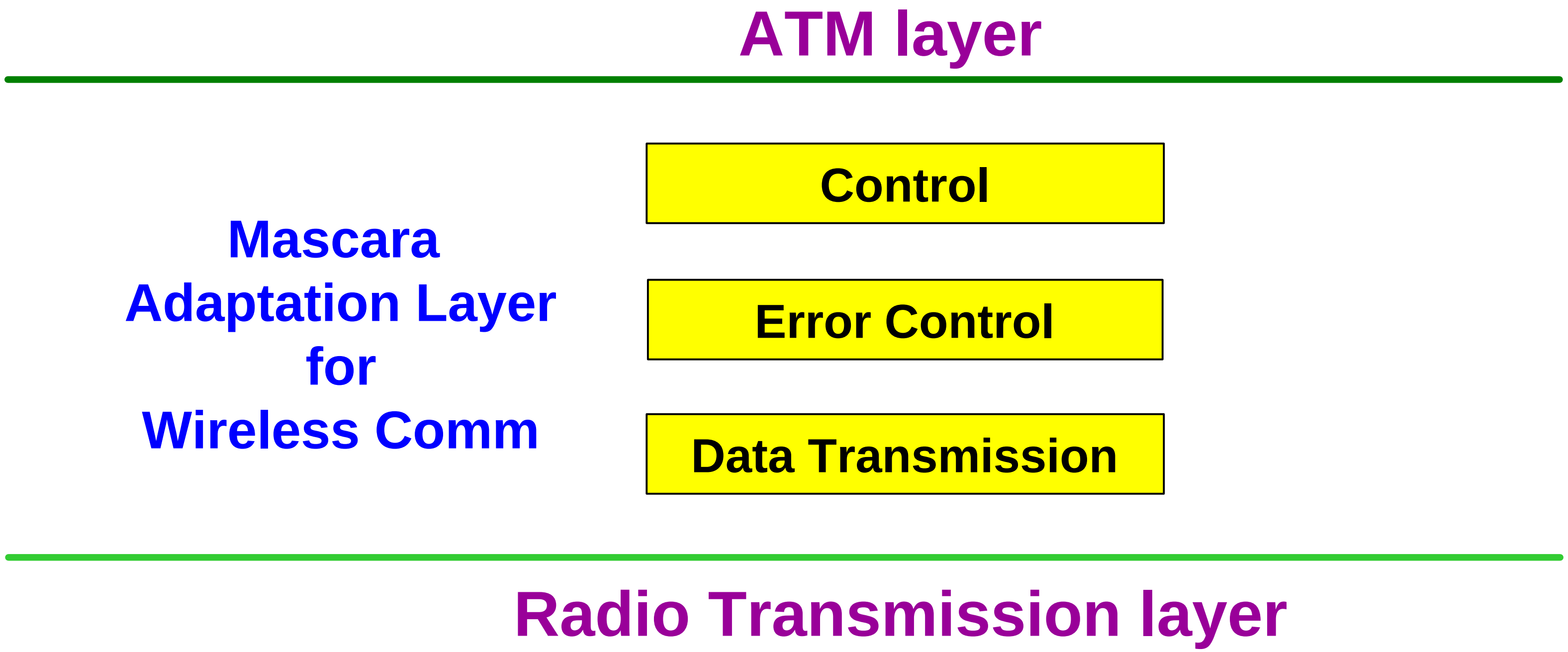
- **E** sends requests **s** only if **x=0**
- **E** responds **res(y)** iff **Sys** has sent **req(x)** and **y < x+5**

Applications

- ATM adaptation layer transport protocol (SSCOP)
  - live analysis, weak bisimulation minimization
  - state size : 2000B → 100B
  - unexplorable → 1 000 000 states
- Medium access for wireless ATM (Mascara)
  - live analysis, slicing, m-calculus checking
- Ariane-5 flight controller (40 minutes of flight)
  - description obtained by reengineering
  - many timers (smallest with 70ms rate)
  - 31 SDL processes

Mascara Protocol

- Verification case study of Esprit-LTR Vires project
- Medium Access Control protocol for wireless ATM
  ⇒ mediation between access points and mobile terminals



ATM layer

Mascara Adaptation Layer for Wireless Comm

Control

Error Control

Data Transmission

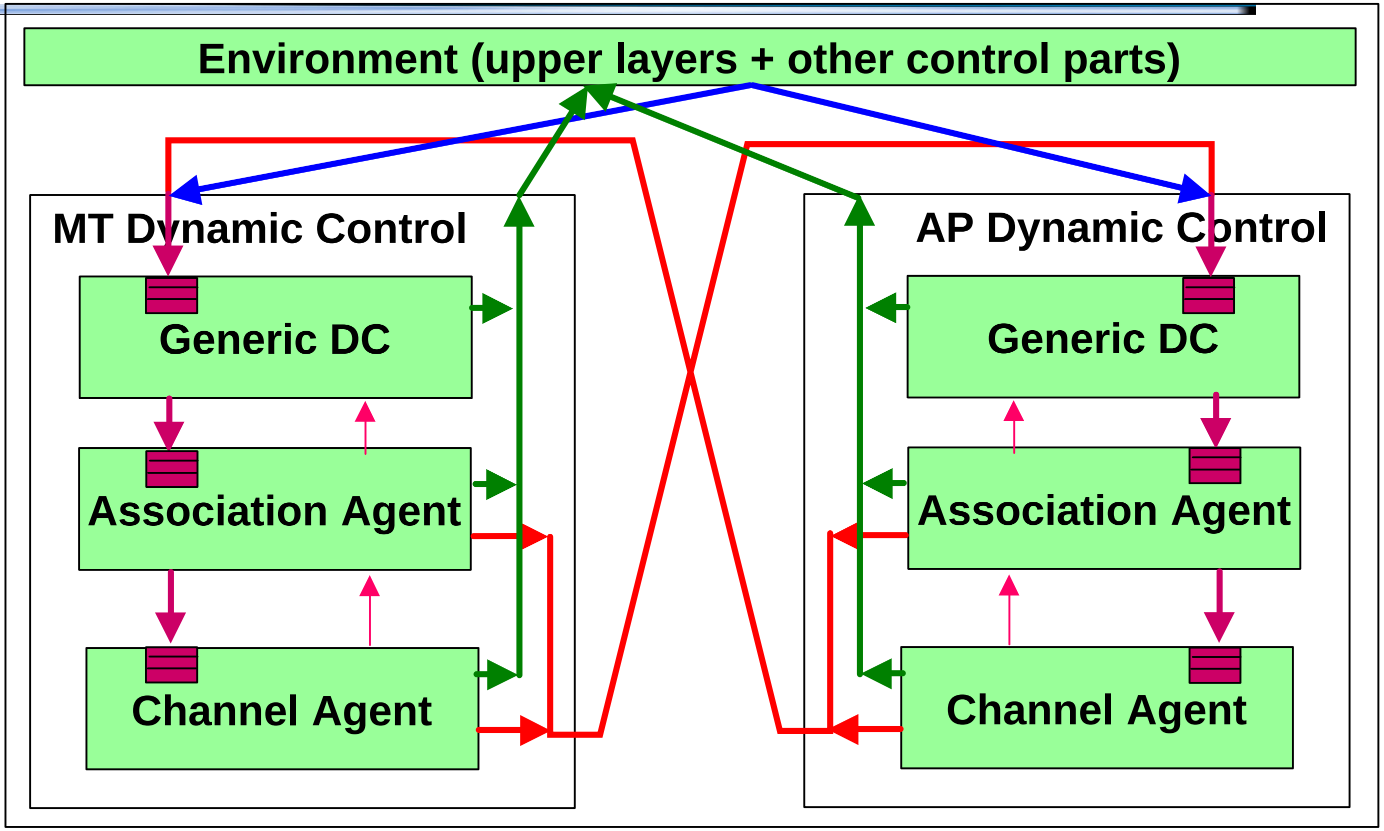
Radio Transmission layer

Mascara Dynamic Control

- **Set up and release associations and virtual connections**

  (address mapping, ressource management)

- **8 SDL processes + environment**



Medium size protocol: 10 000 lines of textual SDL

⇒ complex data structures, large number of messages and potentially interacting protocols

Mascara: modeling choices

Environment and Requirements

1. unrestricted environment → queues of unbounded length

- restrict the number of requests per time unit

2. a priori no functional environment restrictions and no requirements given

- start with simple properties and chaotic environment and strengthen as much as possible/necessary

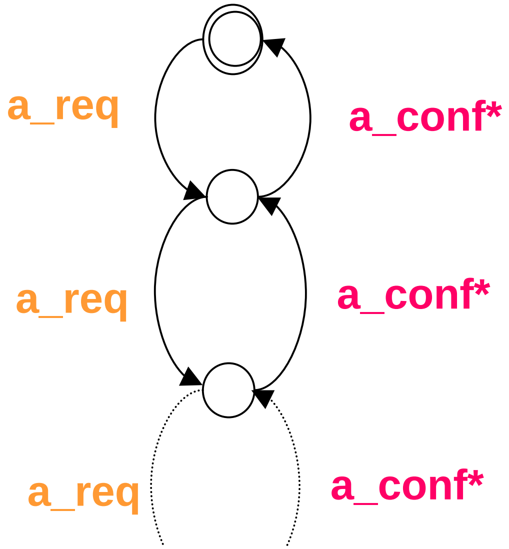
Expression of requirements

- temporal logic
- abstract behaviors in terms of LTS: comparison modulo (bi)simulation or computation of exact property modulo some observation criterion

Expression of Requirements

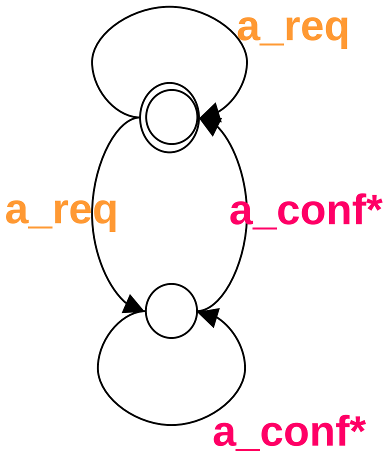
Example: « each association-request will be confirmed »

most general



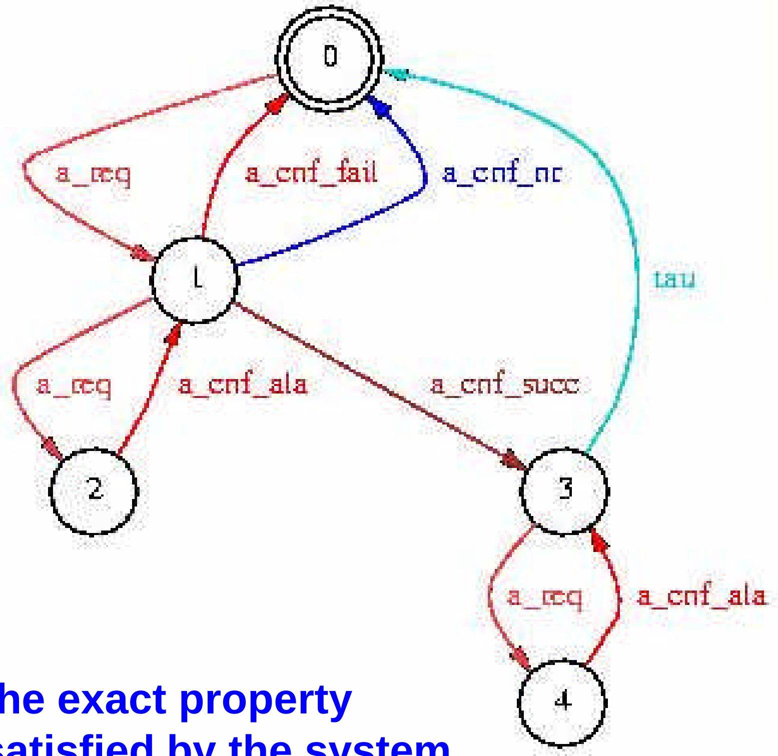
non regular

regular approximation



much weaker

we compute



the exact property satisfied by the system

Mascara: verification strategies

Direct generation failed even using all optimizations

Use of a compositional approach:

- Compositional generation
  - generate and minimize the LTS associated to each process
  - apply parallel composition at the LTS level
- Compositional verification
  - split a global property into a set of local properties
  - verify each local property using an abstract environment

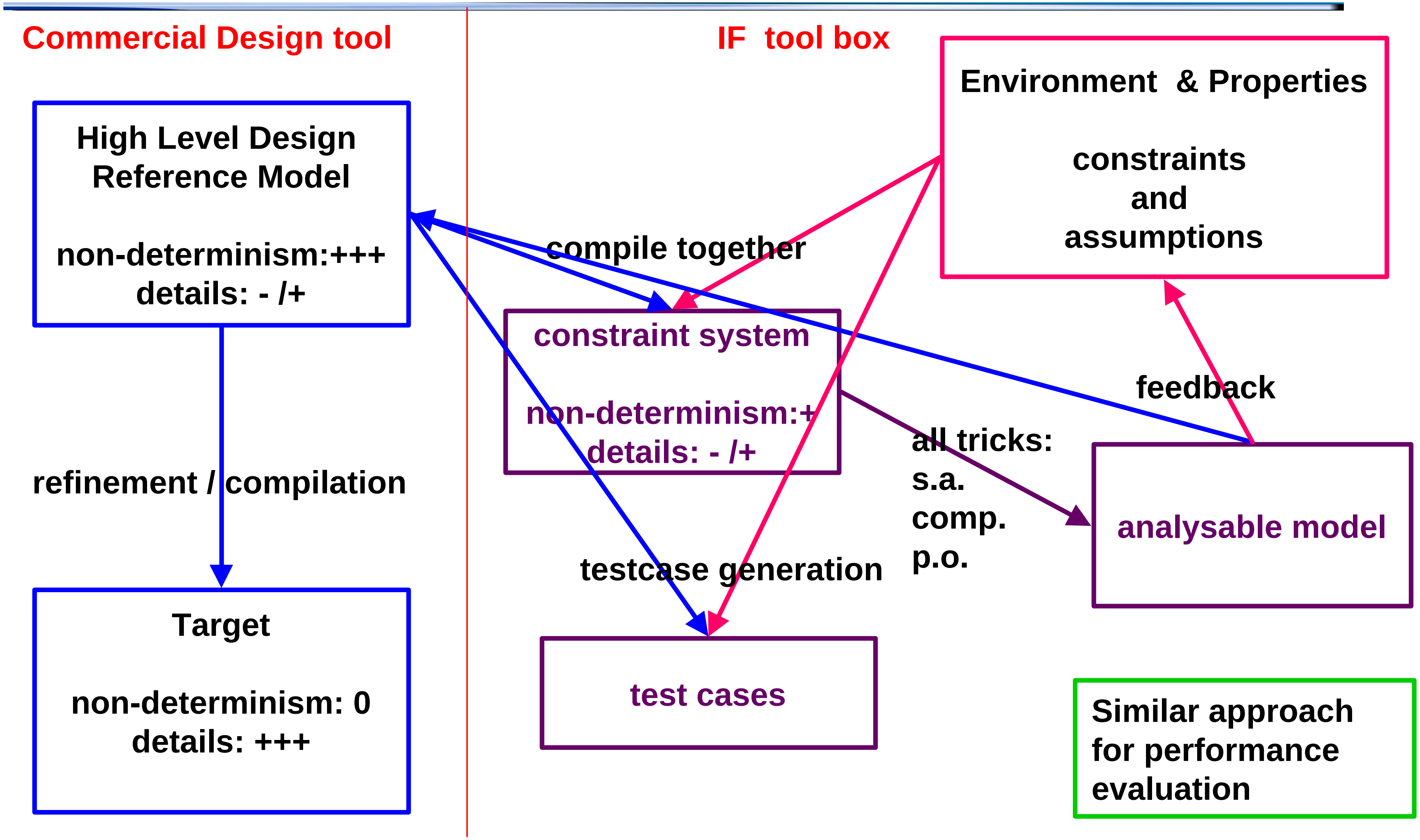
In combination with:

- static analysis
- partial order reduction

Mascara: Complexity results

ent	n°	method	states	reduc	trans	redu	time	redu
AP	1	no reduction	7 000 K	-	30 000 K	-	3h	-
	2	p.o.	900 K	8	1 800 K	17	37m	5
	3	live reduction	400 K	17	1 500 K	20	12m	15
	4	p.o. + live	28 K	250	52 K	577	1m52	118
MT	5	no reduction	4 300 K	-	12 000 K	-	2h51	-
	6	p.o.	3 100 K	1.3	7 400 K	1.6	1h30	1.9
	7	live reduction	63 K	68	325 K	36	1m03	162
	8	p.o. + live	6 K	716	20 K	600	7s	1460
	9	live + po + slice	1 K	4300	3 K	4000	4s	2550
all	10	live + p.o.	--		--		--	
all	11	4_{min} || 8_{min}	218 K		1 140 K		n.a.	

Conclusions: method for design validation



Similar approach for performance evaluation

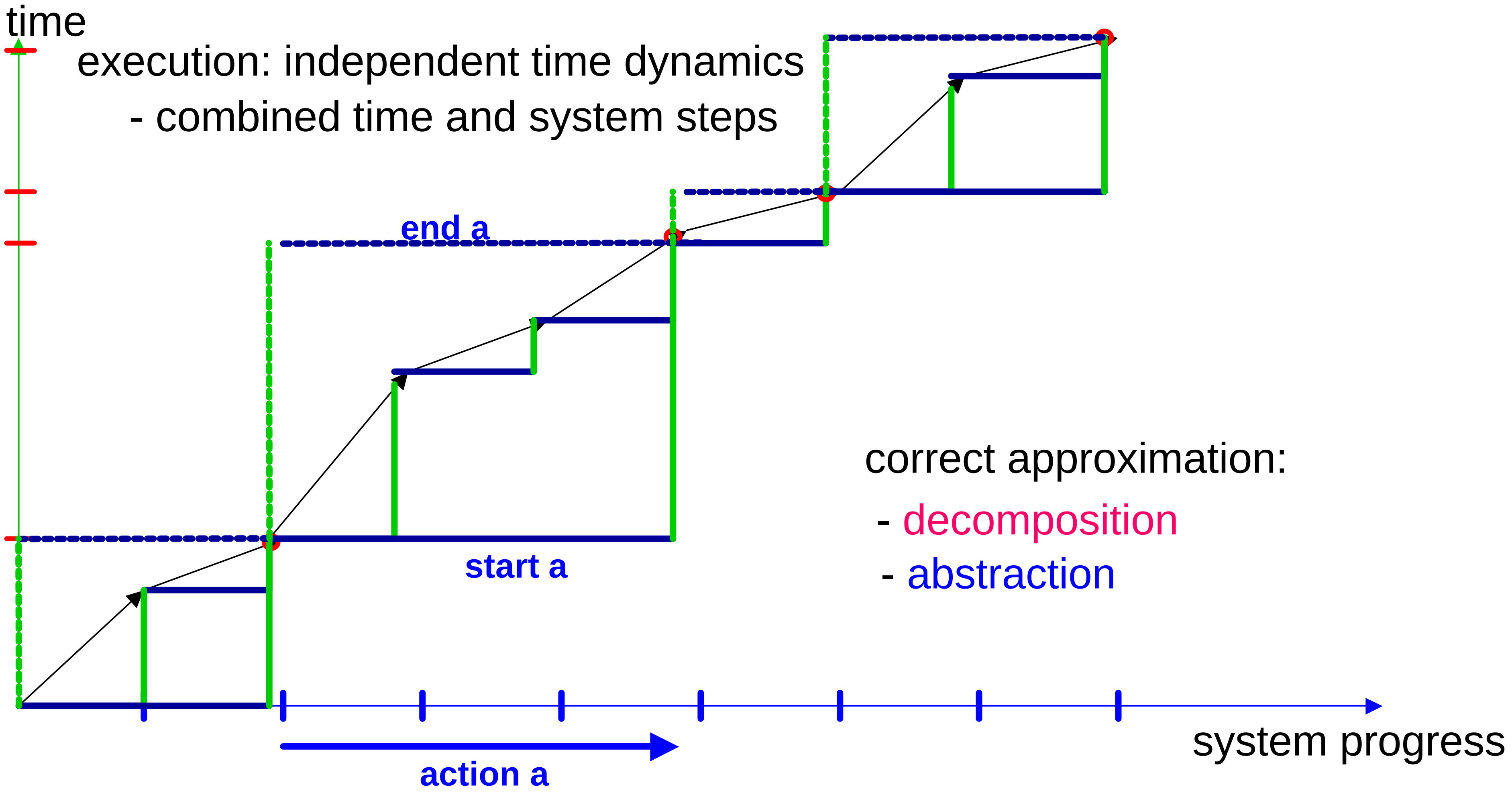
Tool Perspectives

- dynamic features are needed:
  - for connection with UML, JAVA, ...
  - for connection with symbolic validation tools

  definition of dynamicIF
- more general annotations of type assume/assert for requirement expression and test case generation
- more static analysis, abstraction and constraint propagation: connection with PVS based InVest tool
- more compositional verification methods
- better diagnostic facilities
- Connections:
  - connection with ASM tools
  - connection with performance evaluation tools

http://www-verimag.imag.fr/~async

Time and system progress in simulation


time
execution: independent time dynamics
- combined time and system steps
end a
start a
action a
system progress

correct approximation:

- decomposition
- abstraction

Time and system progress in simulation

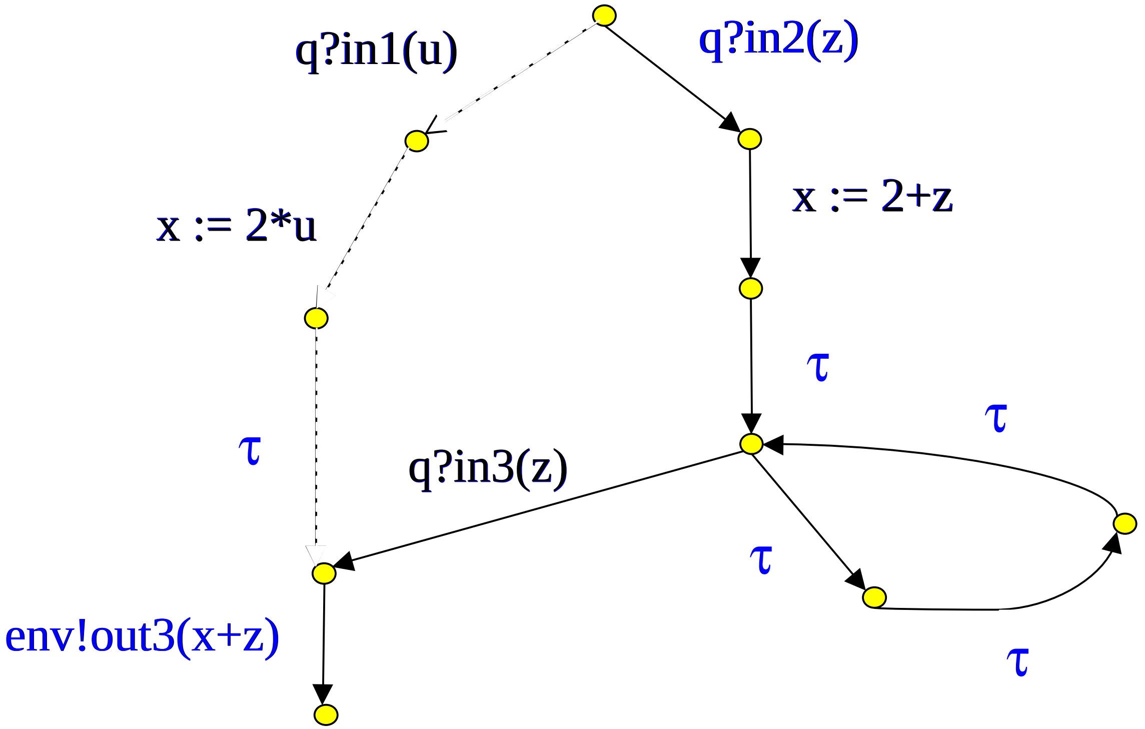
During simulation/validation:

- Problem: how to decide the time point of the next event: now? or should time progress, and how much ?

Time progress must depend on assumptions made by the designer

Slicing: example

var: u,w,x,y,z



Slicing criterion:

- observable events: in2, out3

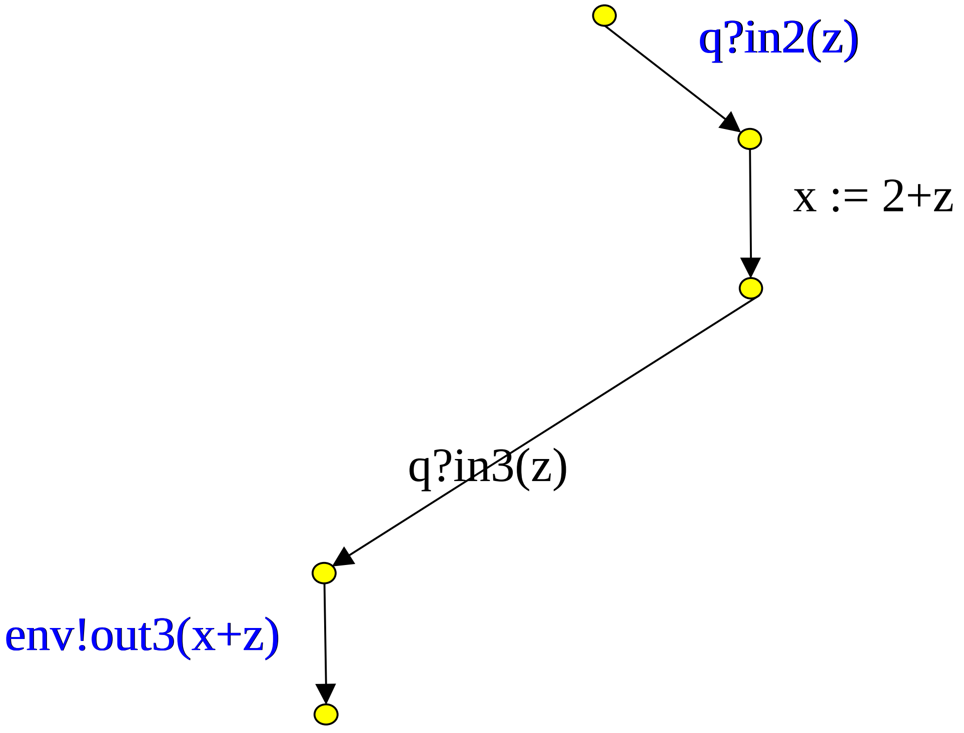
var: u,x,z

- environment: in2, in3, in4

var: x,z

Slicing: example

var: x,z



Slicing criteria:

- observable events: in2, out3

var: u,x,z

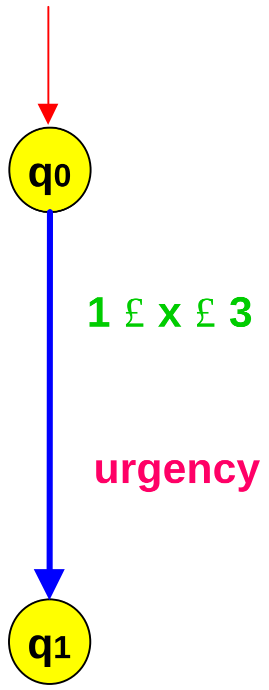
- environment: in2, in3, in4

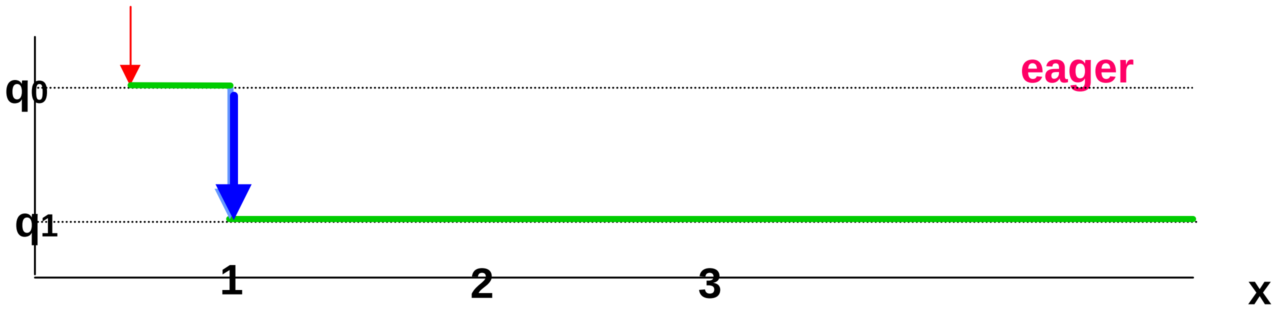
var: x,z

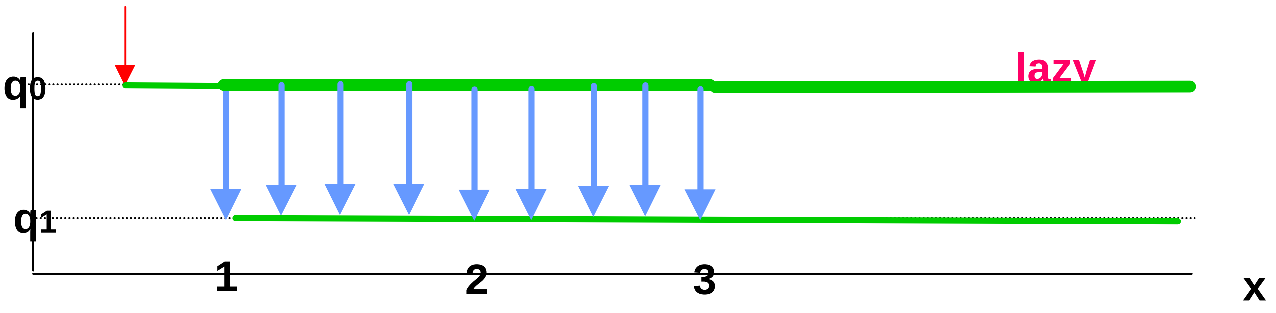
- weak bisimulation reduction

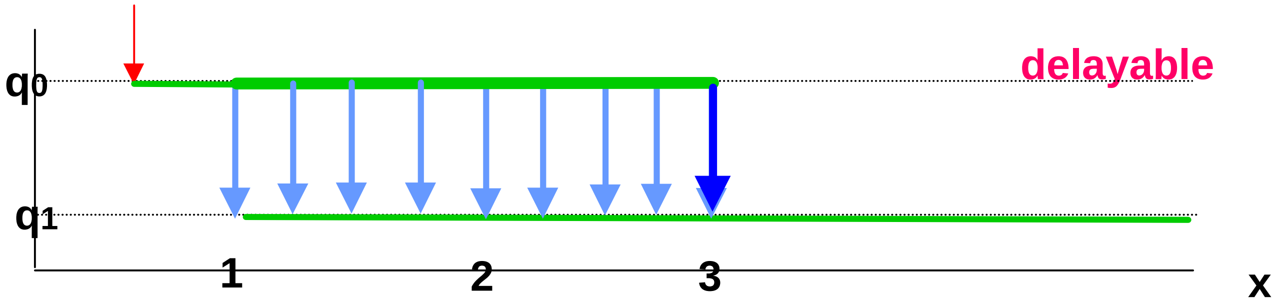
Timed automata with urgency 2

clock x;

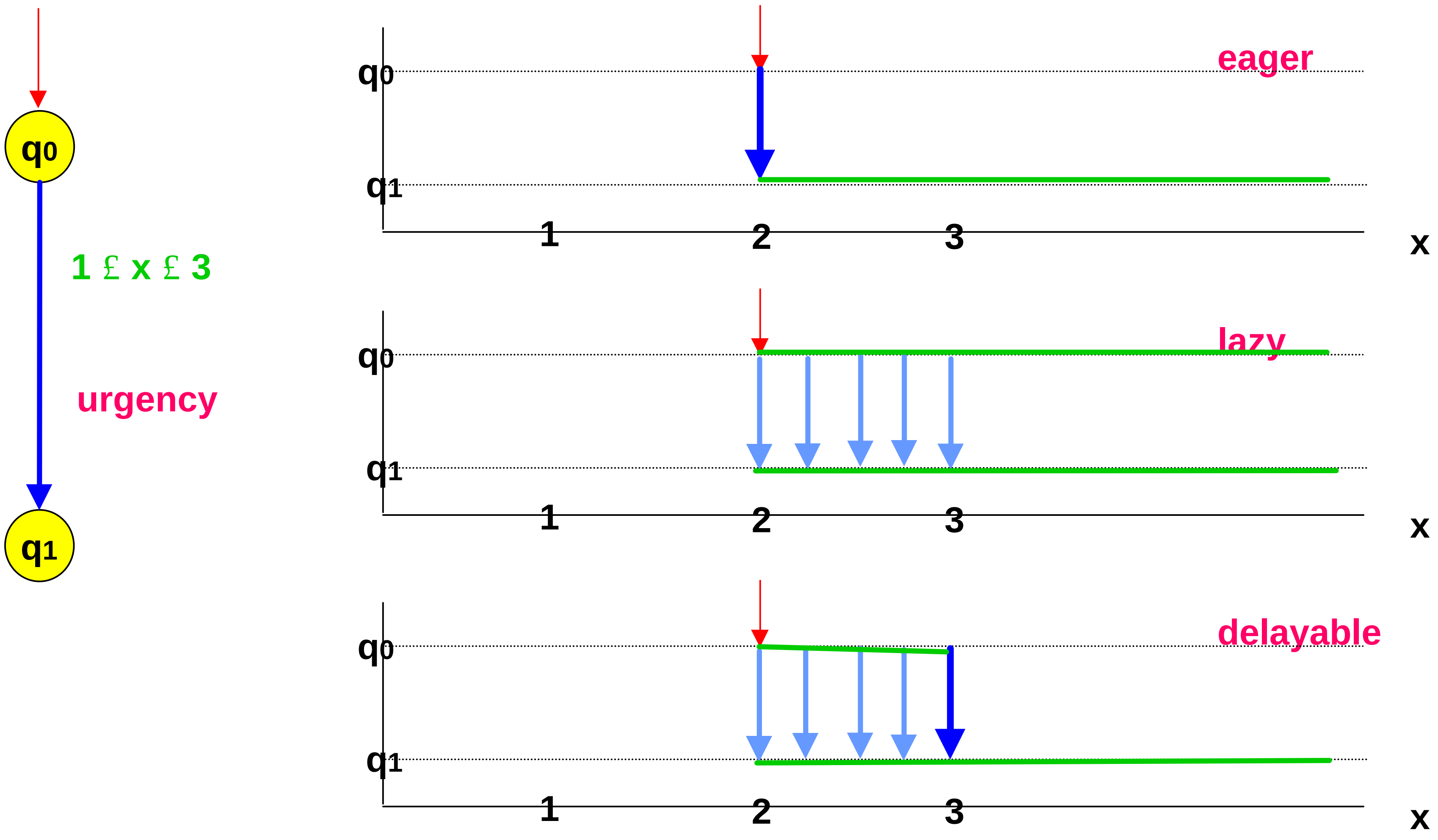




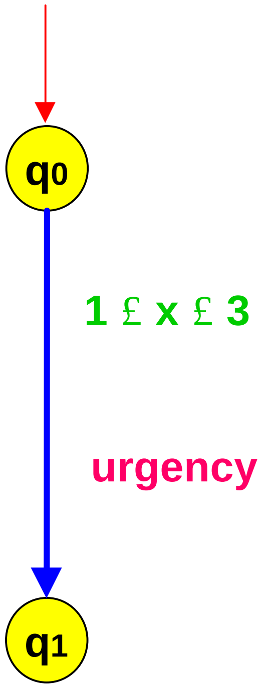


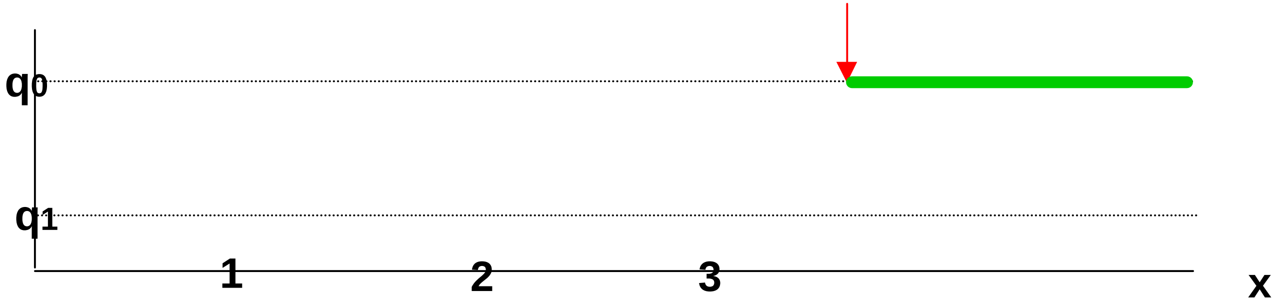


Timed automata with urgency 2

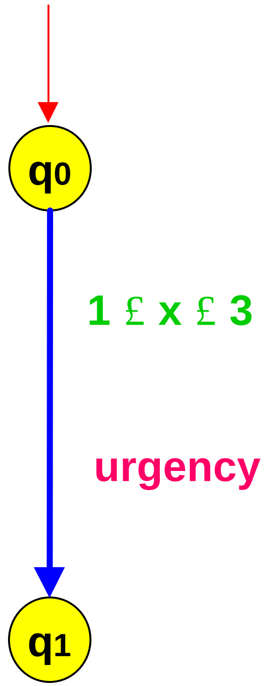


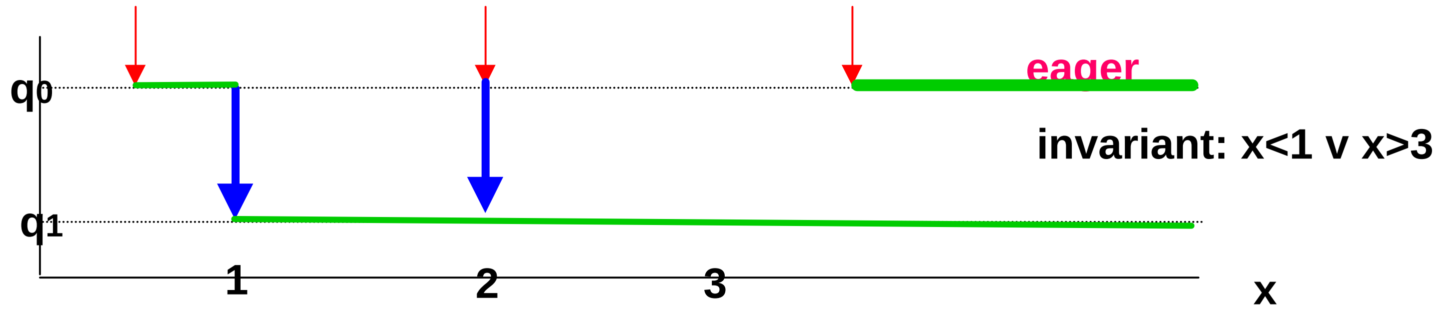
Timed automata with urgency 2





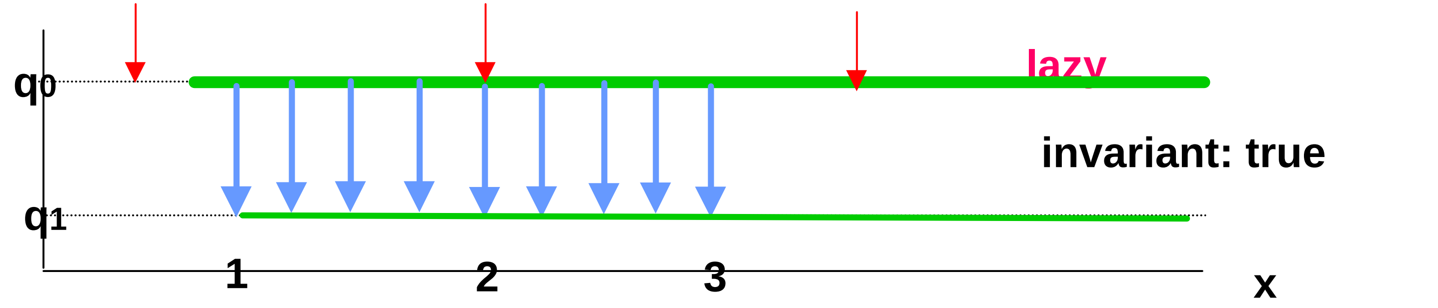
Timed automata with urgency 2





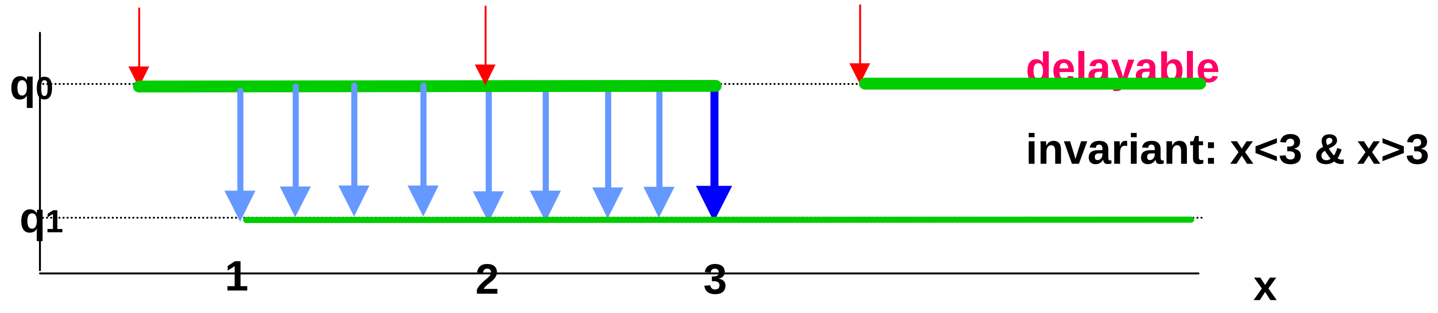
eager

invariant: x<1 v x>3



lazy

invariant: true



delayable

invariant: x<3 & x>3